Parallel MP2-Energy Evaluation: Simulated Shared Memory Approach on Distributed Memory Parallel Machines

AJAY C. LIMAYE*

Interdisciplinary School of Scientific Computing and Department of Chemistry, University of Pune, Pune 411 007, India

Received 6 February 1996; accepted 26 June 1996

ABSTRACT

A parallel algorithm for four-index transformation and MP2 energy evaluation, for distributed memory parallel (MIMD) machines is presented. The underlying serial algorithm for the present parallel effort is the four-index transform. The scheme works through parallelization over AO integrals and, therefore, spreads the $O(n^3)$ memory requirement across the processors, reducing it to $O(n^2)$. In this sense, the scheme superimposes a shared memory architecture onto the distributed memory setup. A detailed analysis of the algorithm is presented for networks with 4, 6, 8, 10, and 12 processors employing a smaller test case of 86 contractions. Model direct MP2 calculations for systems of sizes ranging from 160 to 238 basis functions are reported for 11- and 22-processor networks. A gain of at least 40% and above is observed for the larger systems.

Introduction

With the advent of high-speed computers, post-Hartree–Fock (HF) electronic structure calculations have become routinely possible. Although the workstations of today are able to handle only medium size systems, such types of calculations have become the norm today. The task at hand is to harness the power of several workstations together and make them work in unison on a single problem. With this option of parallel computing now being made available commercially, it has spurred parallel code development in all the fields of science. The area of *ab initio* quantum chemistry too is alight with much parallelization activity. The primary candidate for parallelization in *ab initio* studies of molecules is the SCF

*Author to whom all correspondence should be addressed. E-mail: sakul@unipune.ernet.in

CCC 0192-8651 / 97 / 040552-10

method.[1–5] The post-HF calculations, which are required to process a very large number of electron repulsion integrals, has also seen some interest, even in the past. However, the parallel machines at that time posed severe restrictions on the size of the systems one can handle. Today, as more powerful computers are being made available, parallel efforts in this area have also gained momentum.

Due to both its simplicity (in terms of equations) and complexity (in terms of computational and storage requirements) the four-index transformation problem, which is a major obstacle in all post-HF studies, such as MP series, SD-CI, etc., becomes the prime candidate for parallelization. The four-index transformation problem involves transformation of all atomic orbital (AO)-level electron repulsion integrals to those at the molecular orbital (MO) level. The computational complexity of such an algorithm is typically $O(n^5)$, where n is the number of AOs.[6] The current work presents a parallel four-index transformation algorithm for distributed memory parallel machines.

As a first step in the post-HF venture, one considers the Møller–Plesset (MP) perturbative approach,[7] due to its simplicity and size consistency. For example, the second-order energy correction within this theory for closed-shell systems can be given as:

$$E_{\mathrm{MP2}} = \sum_{i,j}^{\mathbb{NO}} \sum_{a,b}^{\mathbb{NV}} \left[\frac{(ia|jb)^2 - (ia|jb)(ib|ja)}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} \right] \quad (1)$$

where i, j run over occupied molecular orbitals (MOs) $\mathbb{NO}$; a, b run over the virtual set of MOs, $\mathbb{NV}$; and ϵ_i is the orbital energy of orbital i. This study also discusses the use of permutational symmetry of MO-level integrals in the context of MP2-energy evaluation.

The subsequent sections present the sequential as well as parallel algorithms for MP2-energy evaluation and four-index transformation.

Sequential Algorithm

Several algorithms that carry out four-index transformation have been suggested to date.[6,8–15] The set of equations for one of the widely used four-index transformation algorithms with cubic memory requirements is:

Operation	Total memory requirement	
$(iq\|rs) = \sum_{p=1}^{n} (pq\|rs)C_{ip}$	$\mathbb{M} * n * n * (n+1)/2$	(2)
$(ij\|rs) = \sum_{q=1}^{n} (iq\|rs)C_{jq}$	$n * (n+1)/2$	(3)
$(ij\|ks) = \sum_{r=1}^{n} (ij\|rs)C_{kr}$	n	(4)
$(ij\|kl) = \sum_{s=1}^{n} (ij\|ks)C_{ls}$	1	(5)

where i, j, k, l run over the number of MOs; p, q, r, s run over the number of AOs; C is the MO coefficient matrix; n is the number of AOs; and $\mathbb{M}$ is the number of MOs.

For closed-shell MP2-energy evaluation, only the $(ia|jb)$-type integrals are required, where i, j run over an occupied set and a, b run over the virtual set of MOs. In such a case, the loop indices over MOs are suitably modified to generate the required set of MO integrals. The memory requirement is:

Operation	Total memory requirement	
$(iq\|rs) = \sum_{p=1}^{n} (pq\|rs)C_{ip}$	$\mathbb{NO} * n * n * (n+1)/2$	(6)
$(ia\|rs) = \sum_{q=1}^{n} (iq\|rs)C_{aq}$	$n * (n+1)/2$	(7)
$(ia\|js) = \sum_{r=1}^{n} (ia\|rs)C_{jr}$	n	(8)
$(ia\|jb) = \sum_{s=1}^{n} (ia\|js)C_{bs}$	1	(9)

where $\mathbb{NO}$ is the number of occupied MOs. The transformed integrals are required to be stored for evaluation of MP2 energy, which are $\mathbb{NO} * \mathbb{NV} * \mathbb{NV}$ in number, for each occupied orbital i, where $\mathbb{NV}$ is the number of virtual MOs. When the available memory is insufficient, the transformation can be carried out in batches. Here, the loop over the first index (i.e., i) is suitably modified, and the whole procedure is repeated for every batch. If one adheres to the above algorithm, the minimal memory requirement for MP2 energy evaluation (when each

batch consists of one orbital i) is:

$$n*n*(n+1)/2 + n*(n+1)/2 + n + \mathbb{NO}*\mathbb{NV}*\mathbb{NV} \quad (10)$$

Instead of carrying out the transformation in the order as given in eqs. (2)–(5), these are usually carried out in the order:

Operation	Total memory requirement	
$(iq\|rs) = \sum_{p=1}^{n} (pq\|rs)C_{ip}$	$\mathbb{NO}*n*n$ $*(n+1)/2$	(11)
$(iq\|js) = \sum_{r=1}^{n} (iq\|rs)C_{jr}$	$n*n$	(12)
$(ia\|js) = \sum_{q=1}^{n} (iq\|js)C_{aq}$	n	(13)
$(ia\|jb) = \sum_{s=1}^{n} (ia\|js)C_{bs}$	1	(14)

The transformed integrals for MP2-energy evaluation then requires only the $\mathbb{NV}*\mathbb{NV}$ array, since for a given pair (i, j) of occupied orbitals one has integrals over all the pairs (a, b) of virtual orbitals. Therefore, if the MP2-energy evaluation is carried out as given in eqs. (11)–(14), the minimal memory requirement comes down to

$$n*n*(n+1)/2 + n*n + n + \mathbb{NV}*\mathbb{NV} \quad (15)$$

Making use of permutational symmetry in MO-level integrals, the MP2 energy equation for closed-shell systems [see eq. (1)] can be rewritten as:

$$\mathrm{Emp2} = \sum_{i>j}^{\mathbb{NO}} \sum_{a,b}^{\mathbb{NV}} \left[\frac{2(ia|jb)^2 - (ia|jb)(ib|ja)}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} \right] + \sum_{i}^{\mathbb{NO}} \sum_{a,b}^{\mathbb{NV}} \left[\frac{(ia|ib)^2}{\epsilon_i + \epsilon_j - \epsilon_a - \epsilon_b} \right] \quad (16)$$

Although modifying the MP2 equation does not decrease the total memory requirement, it reduces the computational complexity by nearly half. Eq. (16), makes use of permutational symmetry over occupied orbital indices. Further use of permutational symmetry (i.e., the one over virtual orbital indices a and b) requires that all transformed integrals are ready before carrying out the MP2 energy evaluation. In such a case, the energy evaluation cannot be carried out ''on-the-fly.''

Parallel Algorithm

The present work considers a distributed memory parallel architecture for parallel implementation of four-index transform and subsequent MP2-energy evaluation. If the memory restrictions are too severe (such as with a 4-MB per processor) and the communication speeds are at a ''bottleneck,'' one may consider parallelization of the two two-index transform procedure,[16–19] as was done in a previous work.[19] As the memory restrictions become less stringent and the link speeds are reasonably high, one considers the cubic memory algorithm (i.e., the four one-index transform algorithm for parallelization[20–22]). Nielsen et al.[23] have parallelized the direct serial algorithm similar to the MP2 algorithm formulated by Head-Gordon et al.,[14] employing an object-oriented programming language C++. Harrison et al.[24] present an excellent review of various parallelization efforts on four-index transform and *ab initio* molecular electronic structure methods on parallel computers.

One of the quickest and intuitively appealing ways of parallelizing the MP2 integral evaluation algorithm [eqs. (11)–(14)] is to distribute the topmost index among the available processors.[20–22] Thus, the loop over the occupied MO index [i.e., i in eq. (11)] is distributed among the processors. Although it is very intuitive and simple, the scheme has certain lacunae. Because the architecture under consideration is of distributed memory nature, in such a parallel scheme, the processor which has the least memory dictates the size of the molecule. If one could reduce the minimum required memory from $O(n^3)$ to $O(n^2)$, then still bigger jobs would be possible. Moreover, when the number of occupied MOs is not perfectly divisible by the number of processors, the power of the processors with lower number of occupied orbitals remains unutilized for some duration of time.

One would therefore like to find a way to ''smoothen'' this inefficient memory as well as compute resource management. In fact, such a parallel algorithm has already been implemented by Nielsen et al.[23] which requires $O(n*\mathbb{NO})$ memory. Their parallel implementation, however, requires double (for V1 algorithm and four times for V2 algorithm) the total permutationally independent AO integrals. The present parallelization scheme utilizes all permutational symmetries, and yet it restricts the memory requirements down to $O(n^2)$. Rather than distributing the topmost loop

over occupied MOs, the present algorithm distributes the AO-level index among the processors. The minimum $O(n^3)$ memory requirement per processor is thus spread across the processors, thereby reducing it to $O(n^2)$. The present algorithm, in this sense, superimposes a shared-memorylike structure onto the distributed memory setup (in fact, Nielsen et al.[23] also superimposed such a structure). The sequence of operations is slightly modified, as given below, to suit this philosophy.

Operation	
$(iq\|rs) = \sum_{p=1}^{n} (pq\|rs)C_{ip}$	(17)
$(iq\|js) = \sum_{r=1}^{n} (iq\|rs)C_{jr}$	(18)
$(iq\|jb) = \sum_{s=1}^{n} (iq\|js)C_{bs}$	(19)
$(ia\|jb) = \sum_{q=1}^{n} (iq\|jb)C_{aq}$	(20)

The AO integrals required for transformation may either be stored onto disks (conventional calculation) or are re-evaluated each time (direct calculation). Each processor may either evaluate all the nonzero, permutationally independent integrals or a mutually exclusive set of them. In case a mutually exclusive set of AO integrals is evaluated, these integrals need to be communicated to other processors, because every processor requires all integrals to carry out the first one-index transform. Such a redistribution can be carried out in two ways:

1. Each processor broadcasts its own set of integrals to all of the other processors.[22] It also receives integrals from all remaining processors. The drawback of such a scheme is that computation power of one processor is wasted while carrying out the first one-index transform, because each processor remains idle for one session.
2. An alternative is to cycle the integrals among the processors in a ring. In such a scheme no element of computational power remains underutilized.

The above processes (either 1 or 2) are executed until all of the AO integrals are exhausted (until every processor exhausts its share of AO integrals). The one-index transformed integrals are formed at the end of this session.

The present algorithm for parallelization employs the latter scheme for redistributing the AO integrals. In the current approach for parallelization of a four-index transform, rather than distributing (breaking) the loop over i, the loop over q is distributed among the processors, where i is the loop index over occupied MOs and q is the loop index over AOs [see eqs. (17)–(20)]. If one breaks the loop over q onto the available number of processors, then each processor will carry out the first three one-index transforms [eqs. (17)–(19)] for a given subset of AO orbitals, say $\mathbb{QS}$ to $\mathbb{QE}$. Such a distribution is possible because, in the first three transformation steps, the second index, q, is not required in its entirety (i.e., q is not necessarily required to run over all AOs). Once the $\mathbb{QS}$ and $\mathbb{QE}$ is decided for each processor, the first three transformation steps are carried out as follows:

Operation
$(iq\|rs) = \sum_{p=1}^{n} (pq\|rs)C_{ip};\ q = \mathbb{QS}, \mathbb{QE}$
$(iq\|js) = \sum_{r=1}^{n} (iq\|rs)C_{jr};\ q = \mathbb{QS}, \mathbb{QE}$
$(iq\|jb) = \sum_{s=1}^{n} (iq\|js)C_{bs};\ q = \mathbb{QS}, \mathbb{QE}$

The AO integrals required for the first one-index transformation are circumnavigated among the processors in a ring fashion as discussed earlier. As each processor is allowed to handle only a subset of AOs for the second index (i.e., q) between $\mathbb{QS}$ and $\mathbb{QE}$, some of the AO-level integrals may be eliminated at each step in the circulation phase. If all the four indices of an AO integral lie between $\mathbb{QS}$ and $\mathbb{QE}$ for a given processor, then that integral will be of no use to any other processor. The contribution of each AO integral $(pq|rs)$ is evaluated as:

Contribution	One-index transformed MO integral	Subject to the conditions
$(pq\|rs)C_{ip}$	$(iq\|rs)$	$\mathbb{QS} \le q \le \mathbb{QE}$
$(pq\|rs)C_{iq}$	$(ip\|rs)$	$p \cdot \mathrm{ne} \cdot q$ $\mathbb{QS} \le p \le \mathbb{QE}$
$(pq\|rs)C_{ir}$	$(is\|pq)$	$pq \cdot \mathrm{ne} \cdot rs$ $\mathbb{QS} \le s \le \mathbb{QE}$
$(pq\|rs)C_{is}$	$(ir\|pq)$	$r \cdot \mathrm{ne} \cdot s$ & $pq \cdot \mathrm{ne} \cdot rs$ $\mathbb{QS} \le r \le \mathbb{QE}$

where: $p \geq q$; $r \geq s$; i runs over occupied MOs, $\mathbb{NO}$ and $pq = p*(p-1)/2 + q$. At each step (i.e., at every processor) in the circulation phase some of the integrals will be eliminated. Thus, the communication load will be continuously decreasing during this phase. Such a decrease is not possible if the AO integrals are broadcast to all the processors in one shot.

Once the first one-index transformation is over, the next two one-index transforms [eqs. (18) and (19)] are carried out with the one-index transformed integrals available with each processor. No communication is required for these two steps. Each processor handles exactly the same MO index at any given instant. Each processor carries out the fourth one-index transform [eq. (20)] for its subset from $\mathbb{QS}$ to $\mathbb{QE}$ as follows:

$$(ia|jb)' = \sum_{q=\mathbb{QS}}^{\mathbb{QE}} (iq|jb)C_{aq}$$

At this stage, however, as each processor has only a mutually exclusive partial set of q indices, communication among the processors is necessary. Each processor forms an $\mathbb{NV}^2$ array for a given pair (i, j) of occupied MOs. This array contains the semiprocessed integrals $(ia|jb)'$, for every pair (a, b) of virtual MOs and given (i, j). The partial contributions $(ia|jb)'$ are then added to get a fully transformed MO integral $(ia|jb)$. This global sum is carried out in such a way that the communications from various processors are overlapped.

The above parallel algorithm can be expressed as a set of equations for each processor as follows:

The MP2 energy evaluation is then carried out sequentially. Each processor communicates for all the pairs (i, j), $i \geq j$, of occupied MOs. Thus, the total number of communication calls required is $\mathbb{NO}*(\mathbb{NO}+1)/2$, with $\mathbb{NV}*\mathbb{NV}$ array elements per call. The per-processor memory requirement is:

$$(\mathbb{QE} - \mathbb{QS} + 1)*n*(n+1)/2 + n*n + n + \mathbb{NV}*\mathbb{NV}$$

The indices, $\mathbb{QS}$ and $\mathbb{QE}$, can be suitably modified to take into cognizance, the memory inhomogeneity. Thus, the present algorithm reduces the minimal memory requirements to $O(n^2)$, albeit at the cost of increased communication.

The present parallel algorithm has certain similarities with those of Màrquez et al.[22] and Nielsen et al.[23]; however, there are some major differences. Although the underlying serial algorithm is the same, Màrquez et al. have parallelized the four one-index transforms with the first index, i, distributed among the processors, whereas, in the present case the AO index, q, is distributed among the processors. The minimum memory required per processor in the earlier one is $O(n^3)$, whereas in the present case it is $O(n^2)$. Nielsen et al. have parallelized an algorithm similar to that suggested by Head-Gordon et al.[14] The algorithm requires $O(n*\mathbb{NO})$ memory per processor when applied to MP2 integrals. The process will still require $O(n^2)$ memory when implemented for the complete four-index transform. There is no such substantial increase in memory requirements in the present scheme, even if complete four-index transforma-

Operation	Start and end indices for various variables (in each case $q = \mathbb{QS}, \mathbb{QE}$)
$(iq\|rs) = \sum_{p=1}^{n} (pq\|rs)C_{ip}$	$i = 1, \mathbb{NO}$; $r = 1, n$; $s = 1, r$
$(iq\|js) = \sum_{r=1}^{n} (iq\|rs)C_{jr}$	$i = 1, \mathbb{NO}$; $j = 1, i$; $s = 1, n$
$(iq\|jb) = \sum_{s=1}^{n} (iq\|js)C_{bs}$	$i = 1, \mathbb{NO}$; $j = 1, i$; $b = 1, \mathbb{NV}$
$(ia\|jb)' = \sum_{q} (iq\|jb)C_{aq}$	$i = 1, \mathbb{NO}$; $j = 1, i$; $a, b = 1, \mathbb{NV}$
$(ia\|jb) = \sum_{n\text{proc}=1}^{Np} (ia\|jb)$	$i = 1, \mathbb{NO}$; $j = 1, i$; $a, b = 1, \mathbb{NV}$
	(*carried out on one processor*)

tion is performed. In the V1 algorithm suggested by Nielsen et al., however, the parallelization is over AO integrals, since the underlying serial algorithm itself cannot make use of one permutational symmetry, viz. $rs \leq pq$, it leads to a twofold increase in the required AO integrals. In the V2 algorithm which requires significantly less communication, the number of AO integrals required are four times the actual number of permutationally independent AO integrals. In the present case, only permutationally independent AO integrals are necessary. The first communication step circulates the AO integrals for first-quarter transformation in the present algorithm, whereas, it does global addition of partially quarter-transformed integrals in the V1 algorithm of Nielsen et al. Both algorithms require another communication step to bring the required MP2 integrals to the intended processors, i.e., only root processor for the present case, where all the partial integrals are summed up. The present algorithm thus infuses best of both algorithms —thus making use of only permutationally independent integrals (from Màrquez et al.[22]) and reducing the minimum memory requirement by distributing AO integrals (from Nielsen et al.[23]).

Results and Discussion

The present algorithm is implemented on a loosely coupled distributed memory parallel machine, PARAM-9000,[25] which employs Super-Sparc chips as the computation nodes. It contains a network switch, so that the users can view the network as a completely connected one. The configuration used for the present work provides the communication speeds of the order of 2 megabytes per second. The program is written in Fortran using an upcoming standard for parallel message passing systems called Message Passing Interface.[26] The direct four-index transformation and MP2 energy evaluation are carried out for all the test cases reported in the present work.

TABLE I.
Time Required for Circulation of 1 Million Integrals Among the Network of Processors.

No. of Processors	Time (seconds)
2	4.6
4	6.5
6	7.4
8	8.2
10	9.0
12	9.5

Table I depicts the communication delays involved in circulating 1 million integrals (12 bytes per integral) among the processors in the network. It can be observed that although with a greater number of nodes the communication is greater, the time taken by 1 million integrals to visit all processors does not increase linearly with the number of nodes, but saturates with a larger number of nodes.

A detailed analysis of the present parallel implementation is depicted through a smaller test case with *n*-dodecane ($C_{12}H_{26}$) molecule with a STO-3G basis set. The molecule has 86 contractions with 49 occupied and 37 virtual orbitals. The total number of surviving AO integrals is 1.86 million. Table II shows the timings for direct four-index transformation and MP2 energy evaluation employing various numbers of nodes. The testing is carried out employing two versions—A and B. In the version A, the workspace for four-index transformation and MP2-energy evaluation is restricted to 2 lakh elements, whereas in version B it is 1

TABLE II.
Direct Four-Index Transformation and MP2-Energy Evaluation Time for *n*-Dodecane (STO-3G) on Versions A and B (See Text for Details).

	Version A					Version B				
No. of processors	*X*	*Y*	*Z*	Total time	Gain	*X*	*Y*	*Z*	Total time	Gain
4	25	1000	43	1043	1.0	5	263	42	305	1.0
6	17	629	43	672	1.6	3	168	44	212	1.4
8	13	455	44	499	2.1	3	150	45	195	1.6
10	10	396	47	441	2.4	2	121	46	167	1.8
12	9	342	45	387	2.7	2	115	45	160	1.9

X: Number of batches; *Y*: first-quarter transformation timing, in seconds; *Z*: last three-quarter transformation timing, in seconds.

million. The number of batches required for completion of the process is different between versions, which is because the total available workspace is different; the larger the workspace the lower the number of batches. Also, as the number of processors increases, the number of batches decreases. The algorithm works by spreading the $O(n^3)$ memory requirement across the nodes [although each node must have at least $O(n^2)$ memory space]. Hence, an increase in total workspace in the network reduces the required number of batches. Another important aspect brought out from Table II is that the gain is dependent upon the number of batches one has to perform, which in turn depends upon the total workspace available. The fewer the number of batches the higher the time gains. This should be fairly obvious, since one has to perform fewer batches of AO integral evaluations. In version A, therefore, a substantial gain is observed as one migrates from 4 nodes (25 batches) to 12 nodes (9 batches). In the version B, however, such a substantial gain is not observed as there is no large decrease in the number of batches. It can also be seen in version B that, although there is an increase in the number of processors from 6 to 8 and 10 to 12, there are no significant gains in time. This is because the total number of AO integrals circulated around the network remains constant for the pairs, $(6, 8)$ and $(10, 12)$ (the number of batches being same for the pairs). As seen in Table I, with the total communication load remaining constant, an increase in the number of processors results in an increase in time losses. These can be overcome provided the parallel AO-integral evaluation saves enough time. In the present case, since there are only 1.86 million AO integrals, the parallel evaluation time cannot overcome the communication losses.

As seen from Table II, the major compute-and-communication intensive portion of four-index transformation is the first quarter transformation. It consists of an AO-integral evaluation (in the direct case), a one-index transform, and the first communication step, in which all the AO integrals are circulated. The total communication done by a single processor in this first step is the total number of AO integrals. Thus, this step is dictated by the surviving AO integrals. Table III depicts the communication timings for the circulation of 2 million integrals (simulating the dodecane STO-3G case) employing both versions A and B. It can be seen that, as one migrates from 4 nodes to 12 nodes, in both the versions, the communication

TABLE III.
Time Required for Circulating 2 Million Integrals in Batches Among the Network of Processors.

	Version A		Version B	
No. of processors	No. of batches	Total time(s)	No. of batches	Total time(s)
4	25	330	5	68
6	17	228	3	43
8	13	209	3	52
10	10	169	2	38
12	9	149	2	39

timings have *decreased*. This is because the total communication load has decreased with an increase in the number of nodes. With a greater number of processors becoming available, fewer batches need to be performed. Thus, AO integrals are circulated fewer times around the network. In the case of version A, for four nodes, a total of 25 batches are required, each time rotating 2 million integrals. The communication load, therefore, becomes 50 million integrals among the four processors, whereas, for 12 processors, 9 batches are required, resulting into a communication load of 18 million integrals. Also, in version B, there is a substantial decrease in communication load (from 10 million with 4 processors to 4 million on 12 processors). The first communication step, therefore, relies entirely upon the number of AO integrals circumnavigated about the network. Of course, with the same communication load, but increased number of nodes, one does have communication losses, as seen in the case of version B, as well as in Table I. These losses, however, could be dampened if the number of AO integrals to be evaluated is reasonably high. In the present case (version B), however, the smaller number of AO integrals does not compensate for the communication losses.

The last three-quarter transformation and MP2-energy evaluation phase is entirely dominated by the second communication step. This communication step is completely deterministic, in the sense that a fixed number of bytes are transferred at every communication call. The total number of communication calls required in the second communication step is $\mathbb{NO}*(\mathbb{NO}+1)/2$, with $\mathbb{NV}*\mathbb{NV}$ array elements per call. Thus, in all, a total of exactly $\mathbb{NO}*(\mathbb{NO}+1)*\mathbb{NV}*\mathbb{NV}*4$ bytes are transferred to the root processor (8 bytes per element). The communication in this step is also carried out in parallel, by assuming a binary tree

TABLE IV.
Total Direct Four-Index Transformation and MP2-Energy Evaluation Time for Model Systems (in Seconds) Employing Version C (See Text for Details).

Molecule (basis set, basis functions, occupied orbitals)	SCF energy	MP2 energy	AO integrals (in millions)	Total time[a]		Speedup[b]
				11 nodes	22 Nodes	
$C_{12}H_{26}$ (6-31G, 160, 49)	−469.374	−1.126	20	2298	1841	1.24
$C_{13}H_{25}O_2^-$ (6-31G, 185, 60)	−656.315	−1.459	28	4260	3557	1.20
$C_{10}H_8$ (6-31G**, 190, 34)	−383.363	−1.363	43	4696	3313	1.42
$C_{12}H_{26}$ (6-31G*, 232, 49)	−469.567	−1.638	74	15737	9602	1.64
$C_{10}H_8$ [6-31G^{++}(d, p), 238, 34]	−383.375	−1.376	144	18190	13191	1.40

[a] Total time for four-index transformation and MP2-energy evaluation in direct mode.
[b] Speedup = $\frac{\text{11 nodes timing}}{\text{22 nodes timing}}$.

topology, in this phase. Thus, the addition and transmission of semitransformed integrals is carried out in parallel. The final results are added to the root processor, which carries out sequential MP2-energy evaluation. Such a communication pattern yields an almost constant timing for the last three-quarter transformation phase, which is shown in Tables II and V. Observe that neither increased number of nodes nor different versions affect the timings drastically.

Tables I, II, and III serve to show that there will be a stepwise growth in gains as one increases the number of processors in the network. With an increase in the number of processors, the total available workspace also increases, resulting in fewer batches.

Tables IV and V present a variety of test cases employing version C, which has a 4.5-million-element workspace for four-index transformation and MP2-energy evaluation. The model systems cho-

TABLE V.
Detailed Breakup of Direct Four-Index Transformation and MP2-Energy Evaluation Time for Model Systems (in Seconds) Employing Version C.

Molecule (basis functions, occupied orbitals)	11 Nodes			22 Nodes		
		Transformation timings			Transformation timings	
	Number of batches	First index	Last three indices[a]	Number of batches	First index	Last three indices[a]
$C_{12}H_{26}$ (160, 49)	3	1944	354	2	1458	383
$C_{13}H_{25}O_2^-$ (185, 60)	4	3577	683	3	2900	657
$C_{10}H_8$ (190, 34)	3	4399	297	2	3008	305
$C_{12}H_{26}$ (232, 49)	7	14811	926	4	8663	939
$C_{10}H_8$ (238, 34)	5	17560	630	3	12548	643

[a] Including MP2-energy evaluation time.

sen include compact molecules, such as naphthalene ($C_{10}H_8$), and long-chain systems, such as *n*-dodecane ($C_{12}H_{26}$) and *n*-docylcarboxylate ($C_{13}H_{25}O_2^-$). The basis sets chosen are 6-31G, 6-31G*, 6-31G**, and 6-31G^{++}(*d*, *p*). The number of basis functions for the test cases range from the smallest, dodecane (6-31G), to largest, naphthalene [6-31G^{++}(*d*, *p*)], with 160 and 238 basis functions, respectively. Table IV reports the total four-index transformation and MP2-energy evaluation timings on 11 and 22 nodes. Table V gives a detailed breakdown in timing for the entire MP2-energy evaluation process. An important aspect to be noticed from Table V is that the first phase completely dominates the total process. The last three-quarter transformations and MP2-energy evaluation take up less than 20% of the total time. One can observe from the tables that the reductions in time (or speedup) obtained by doubling the number of processors (from 11 to 22 nodes) increases with the size of the system. From Table IV one can see that the speedup ranges from 1.2 for smaller systems to 1.6 for the larger ones. For larger (190 basis functions and above) systems, such a gain in time is appreciable (more than 40%). But, in the case of smaller ones, communication overheads have a dampening effect on the computational gain. In these cases, as seen from Table V, for 22 nodes, the reduction in terms of number of batches is not substantial enough (just an extra batch). Moreover, the number of AO integrals is also reasonably small. The time gains are largest for *n*-dodecane 6-31G*, when the reduction in number of batches is the greatest. The timings in Table V re-emphasize that the speedups for the present parallel algorithm are proportional to the decrease in the number of batches.

In summary, the first phase of the process (first one-index transform) swamps the second phase (the last three one-index transforms and MP2-energy evaluation). The communication overheads depend upon the total memory available in the network. One may completely eliminate the circulation of AO integrals as done by Nielsen et al.,[23] by leaving out one of the AO-integral symmetries, namely $pq \geq rs$. This will result in doubling the AO-integral evaluation, as opposed to four times in the case of Nielsen et al. It is also envisaged that the communication overheads will taper off with the continued advances in communication technology. The same parallel implementation can be employed for full four-index transformation, with the appropriate changes in loop indices and a slight increase in array sizes. The present parallel algorithm makes it possible to take on larger molecular systems by spreading the $O(n^3)$ memory requirement across the network, so that the total memory in the network is seen as a whole, thereby superimposing a shared memory structure onto the distributed memory architecture.

Acknowledgments

The author is grateful to the C-DAC, Pune, for financial assistance and making available the PARAM-9000 machine for use. Special thanks are due to Professor S. R. Gadre and Dr. S. A. Kulkarni for useful discussions and to the referees for their critical comments. Financial support from Department of Science and Technology (SR/SY/C-06/92) to S.A.K. is gratefully acknowledged.

References

1. R. Ernenwein, M. M. Rohmer, and M. Bénard, *Comp. Phys. Commun.*, **58**, 305 (1990).
2. M. W. Feyereisen, R. A. Kendall, J. Nichols, D. Dame, and J. T. Golab, *J. Comput. Chem.*, **14**, 818 (1993).
3. M. E. Colvin, C. L. Janssen, R. A. Whiteside, and C. H. Tong, *Theor. Chim. Acta*, **84**, 301 (1993).
4. R. N. Shirsat, A. C. Limaye, and S. R. Gadre, *J. Comput. Chem.*, **14**, 445 (1993).
5. T. R. Furlani and H. F. King, *J. Comput. Chem.*, **16**, 91 (1995).
6. S. Wilson, Ed., *Methods in Computational Chemistry*, Plenum Press, New York, 1987, vol. 1.
7. C. Möller and M. S. Plesset, *Phys. Rev.*, **46**, 618 (1934).
8. R. K. Nesbet, *Rev. Mod. Phys.*, **35**, 552 (1963).
9. K. C. Tang and C. Edmiston, *J. Chem. Phys.*, **52**, 997 (1970).
10. C. F. Bender, *J. Comput. Phys.*, **9**, 547 (1972).
11. V. R. Saunders and J. H. van Lenthe, *Molec. Phys.*, **48**, 923 (1983).
12. G. H. F. Diercksen, *Theor. Chim. Acta* (Berlin), **33**, 1 (1974).
13. S. Saebo and J. Almlöf, University of Minnesota Supercomputer Institute preprint, UMSI 88/84 (August 1988).
14. M. Head-Gordon, J. A. Pople, and M. J. Frisch, *Chem. Phys. Lett.*, **153**, 503 (1988).
15. M. J. Frisch, M. Head-Gordon, and J. A. Pople, *Chem. Phys. Lett.*, **166**, 281 (1990).
16. R. A. Whiteside, J. S. Binkley, M. E. Colvin, and H. F. Schaefer III, *J. Chem. Phys.*, **86**, 2185 (1987).
17. M. E. Colvin, R. A. Whiteside, and H. F. Schaefer III, *Methods in Computational Chemistry*, S. Wilson, Ed., Plenum Press, New York, 1989.
18. L. A. Covick and K. M. Sando, *J. Comput. Chem.*, **11**, 1151 (1990).

19. A. C. Limaye and S. R. Gadre, *J. Chem. Phys.*, **100**, 1303 (1994).
20. A. D. Bhusari, V. V. Bhate, and S. Pal, *Curr. Sci.* (India), **62**, 293 (1992).
21. R. Wiest, J. Demuynck, M. Bénard, M. M. Rohmer, and R. Ernenwein, *Comput. Phys. Commun.*, **62**, 107 (1991).
22. A. M. Márquez and M. Dupuis, *J. Comput. Chem.*, **16**, 395 (1995).
23. I. M. B. Nielsen and E. T. Seidl, *J. Comput. Chem.*, **16**, 1301 (1995).
24. R. J. Harrison and R. Shephard, *Annu. Rev. Phys. Chem.*, **45**, 623 (1994).
25. P. R. Eknath, A. Degwekar, S. Wandhekar, T. S. N. Murthy, G. Reddy, B. C. Shivakumar, T. R. Arvind, N. V. Narayanan, S. Hussain, Geetanjali, Y. Abhyankar, K. Ganesh, S. Kolhatkar, and A. Karandikar, Param 9000: *Towards a National High-Performance Information Infrastructure*, V. Bhatkar and A. Karandikar, Eds., Center for Development of Advanced Computing (C-DAC) report, 1995.
26. D. W. Walker, *Parallel Comput.*, **20**, 657 (1994).